InTimeFix: A Low-Cost and Scalable Technique for In-Situ Timing Error Masking in Logic Circuits

Feng Yuan and Qiang Xu
CUhk REliable Computing Laboratory (CURE)
Department of Computer Science & Engineering
The Chinese University of Hong Kong, Shatin, N.T., Hong Kong
Email: {fyuan,qxu}@cse.cuhk.edu.hk

ABSTRACT

With technology scaling, integrated circuits (ICs) suffer from increasing process, voltage, and temperature (PVT) variations and adverse aging effects. In most cases, these reliability threats manifest themselves as timing errors on critical speed-paths of the circuit, if a large design guard band is not reserved. This work presents a novel in-situ *timing error masking technique, namely* InTimeFix*, by introducing fine-grained redundant* approximation circuit *into the design to provide more timing slack for speed-paths. The synthesis of the redundant circuit relies on simple structural analysis of the original circuit, which is easily scalable to large IC designs. Experimental results show that* InTimeFix *significantly increases circuit timing slack with low area/power cost.*

1. INTRODUCTION

With technology scaling, integrated circuits (ICs) suffer from increasing process, voltage, and temparature (PVT) variations and aging effects [1]. In most cases, these reliability threats manifest themselves as timing errors on circuit speed-paths (i.e., critical paths that determine circuit speed) [2, 3]. Conventional design methodology requires designers to embed a large design guardband to prevent any timing failure. This conservative design methodology, however, inevitably diminishes the benefit of technology scaling [4]. Consequently, there is a growing research interest to achieve online timing error resilience.

1.1 Related Work

In order to achieve timing error resilience, we can either predict the error occurrence and take proactive actions to avoid them or detect and correct timing errors (or their effects) when they occur. Generally speaking, timing error prediction techniques (e.g., [5]) are applicable to detect gradual increase of circuit delay resulting from aging effects only. In this work, we focus on the more general timing error detection and correction techniques presented in the literature.

Timing Error Recovery: Most existing solutions for timing error resilience try to restore the state of the system to a known-good pre-error state. For example, *RAZOR* [6] implemented such a recovery scheme with microarchitectural support. In this technique, those flip-flops that are driven by speed-paths (denoted as *suspicious FFs* in this paper), are replaced with the so-called *RAZOR-FFs*, which contains a main flip-flop, an additional shadow latch and some control logic. The main flip-flop latches the output signal at the clock edge with possible timing error, while the shadow latch, controlled by a delayed clock signal, latches the signal a fraction of a cycle later, which guarantees to receive the correct value. Consequently, when the shadow and the main FF values do not agree, indicated by the comparator, the timing error is detected. By flushing the pipeline and replaying failed instructions at lower clock frequency, the processor operates correctly with little performance penalty. Bowman *et al.* [7] implemented two novel metastability-immune timing error detectors (namely *error-detection sequential* in their work) and the corresponding error-recovery circuits in an Intel test chip. By removing design guard band used to guarantee "always correct" operations, the above techniques enable *better than worst-case designs* that are more energy-efficient [4], and has inspired a large amount of later research work (e.g., [8–12]).

Timing Error Masking: While *RAZOR*-like techniques are very effective for timing error correction (TEC) in microprocessor datapath with the help of instruction replay, they are very difficult, if not impossible, to be applied to general logic circuits, due to the high cost to checkpoint error-free states in them. It is therefore imperative to develop *in-situ* timing error correction techniques that are able to mask errors without any rollback. There are a few such techniques presented in the literature and they can be classified into two categories: *temporal error masking* and *logic error masking*.

Temporal error masking techniques replace suspicious FFs with sequential circuit elements having time-borrowing capability (e.g., soft-edge flip-flops) and correct timing errors by delaying the arrival time of the correct data to the next logic level (e.g., [13–15]). While effective in many cases, such time-borrowing techniques have the inherent weakness of error effect propagation. That is, even if a suspicious FF can borrow some time from its successive logic level, the timing slack of this level is reduced and hence some initially non-suspicious flip-flops in this level may become suspicious ones and need to be replaced by sequential elements with time-borrowing capability again. Due to this timing error propagation effect, the hardware cost for such temporal error masking techniques can be quite high.

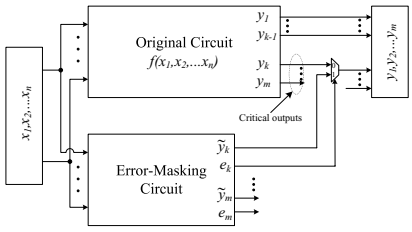


Figure 1: Timing Error Masking Scheme in [16].

In [16], Choudhury and Mohanram proposed to add a redundant logic block to predict the outputs of the circuit upon application of inputs that sensitize speed-paths. With this exact sensitization constraint, the *error-masking circuit* tends to have more timing slack when compared to the original circuit, and hence is immune to timing errors. As shown in Fig. 1, targeting those timing-critical outputs $y_k, ..., y_m$, error masking circuit generates two outputs for each of them, e.g., $\tilde{y}_k$ and e_k for y_k with potential timing error. To be specific, when a speed-path driving y_k is sensitized, e_k is set correspondingly and the original circuit's output y_k is substituted with the fast predicted value $\tilde{y}_k$ to achieve timing error resilience.

DAC'2013, May 29 - June 07 2013, Austin, TX, USA.

1.2 Motivation and Summary of Contributions

The idea of adding redundant logic into the original circuit to mask timing errors on speed-paths as shown in [16] is quite interesting. The main limitation of this work lies in the difficulty of error-masking circuit generation. To be specific, [16] requires to synthesize the so-called *speed-path characteristic function* that represents the set of *all* speed-path activation patterns, which is only practical for small circuit blocks. The associated area/power overhead is also quite significant, as demonstrated in their experimental results.

In this paper, we propose a novel *in-situ* timing error correction technique, namely *InTimeFix*. Unlike [16] that adds *coarse-grained* redundancy by synthesizing the Boolean function that represents the activation of all speed-paths for timing error correction, the redundant TEC circuit in *InTimeFix* is generated at a much more *fine-grained* manner based on the concept of *approximation circuit*, which can be obtained by simple structural analysis of the original circuit. The proposed solution is therefore of low cost and is easily scalable to large IC designs. The main contributions of this paper include:

- we present a novel technique to add redundant approximation circuit into the original design to create a logically-equivalent yet timing-improved circuit, and prove its correctness;
- we propose a low-cost and scalable technique to synthesize timing error masking logic based on simple structural analysis, without necessarily acquiring characteristic function for speed-paths;

From another perspective, *InTimeFix* can be also regarded as a circuit timing optimization technique, since it facilitates to improve circuit timing slack with low hardware cost, as demonstrated in our experimental results. It is also important to emphasize that, as a redundancy scheme, *InTimeFix* is compatible with other timing/power optimization techniques such as gate sizing [19] and dual V_{th} allocation [20], and in fact, these techniques can be combined to further improve circuit performance under variation.

The remainder of this paper is organized as follows. In Section 2 and Section 3, we detail the proposed *InTimeFix* technique for *in-situ* correction of timing errors on speed-paths. Experimental results on various benchmark circuits are then presented in Section 4. We discuss the limitations of *InTimeFix* in Section 5. Finally, Section 6 concludes this paper.

2. IN-SITU TIMING ERROR MASKING WITH APPROXIMATE LOGIC

The concept of *approximation circuit* was first presented in [17], which tries to increase a microprocessor's clock frequency by replacing a complete logic function with a simplified circuit that mimics the function and uses rough calculations to speculate results. In [18], the authors used approximation circuit for concurrent error detection (for random error detection istead of timing error detection) and proposed a methodology to tradeoff error detection capability and area overhead.

In [18], the authors defined *approximate logic* as: Given two Boolean functions F and G, $G0$ is a *0-approximate logic* of F if $G0 = 0 \Rightarrow F = 0$. Similarly, $G1$ is a *1-approximate logic* of F if $G1 = 1 \Rightarrow F = 1$. According to counter-positive law, we can further obtain two statements that are $F = 1 \Rightarrow G0 = 1$ and $F = 0 \Rightarrow G1 = 0$. Consider a Boolean function $F = a + b + \bar{a}cd$, there are 13 on-set minterms[1] and 3 off-set minterms[2] in its truth table. Similarly, a 0-approximate logic function of F, $G0 = a + b + c$, covers 2 out of 3 off-set minterms of F. Generally speaking, since the approximate logic circuit is much simpler when compared to the original circuit, its computational latency is smaller. The basic idea of the proposed *InTimeFix* technique is to generate approximate logic for the original logic function of suspicious FFs in such manner that it covers all the logic minterms that sensitize speed-paths.

[1]The on-set minterm is the minterm where the function outputs logic '1'.

[2]The off-set minterm is the minterm where the function outputs logic '0'.

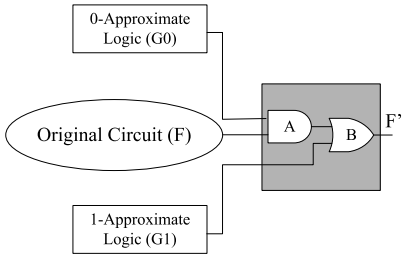


Figure 2: Equivalent Circuit with Approximate Logic.

2.1 Equivalent Circuit Construction with Approximate Logic

Given a logic circuit whose Boolean function is F, suppose $G0$ is a 0-approximate logic for F and $G1$ is a 1-approximate logic for F. we define: P is all the minterms in F's truth table, $P0$ is the off-set minterms of F covered by $G0$ and $P1$ is the on-set minterms of F covered by $G1$. Now, let us construct a circuit $F' = F \cdot G0 + G1$ as shown in Fig. 2. We define $d_{G0}(P0)$, $d_{G1}(P0)$ and $d_F(P0)$ as the worst-case delay among all minterms in P0 through circuit $G0$, $G1$ and F, respectively. Similarly, $d_{G0}(P1)$, $d_{G1}(P1)$, $d_F(P1)$ and $d_F(P-P0-P1)$ are defined. d_{AB} is the total propagation delay of *AND* gate A and *OR* gate B. Assuming approximate logic $G0$ and $G1$ are implemented with simpler logic structures when compared to original circuit F and hence has less computation latency (e.g., $d_{G0}(P0)<d_F(P0)$, $d_{G1}(P0)<d_F(P0)$ and $d_{G1}(P1)<d_F(P1)$), we have the following theorem:

THEOREM 1. *The circuit shown in Fig. 2, $F' = F \cdot G0 + G1$, is logically-equivalent to the original circuit F, and its worst-case timing delay is* $\max\{d_F(P-P0-P1), d_{G0}(P0), d_{G1}(P0), d_{G1}(P1)\} + d_{AB}$.

PROOF. When the original circuit F outputs 1, by applying counter-positive law, its 0-approximate logic must also output 1 (i.e., $G0 = 1$), and hence $F' = 1$. Similarly, when the original circuit F outputs 0, by applying counter-positive law, its 1-approximate logic must also output 0 (e.g., $G1 = 0$), and hence $F' = 0$. Consequently, F and F' are logically-equivalent.

To obtain the worst-case delay for this equivalent circuit F', let us consider the circuit delay before the shaded logic block in Fig. 2 for the following three cases, corresponding to the application of inputs belonging to different set of minterms of the truth table of F/F'.

- When the inputs applied to the circuit belong to $P0$, $G0$ outputs controlling value 0 for *AND* gate A after $d_{G0}(P0)$ and dominates the path through the original circuit F with longer delays. The worst-case delay would be $\max\{d_{G0}(P0), d_{G1}(P0)\}$. Note that $d_{G1}(P0)$ is the time spent to settle $G1$ to be non-controlling value 0 for *OR* gate B.
- When the inputs applied to the circuit belong to $P1$, $G1$ outputs controlling value 1 for *OR* gate B after $d_{G1}(P1)$ and it dominates the path through the original circuit F and $G0$. The worst-case delay in this case is therefore simply $d_{G1}(P1)$.
- When the inputs applied to the circuit belong to $P-P0-P1$, we have to wait for the original circuit F to settle down, and hence the worst-case delay would be $d_F(P-P0-P1)$.

The worst-case timing delay for circuit F' is therefore $\max\{d_F(P-P0-P1), d_{G0}(P0), d_{G1}(P0), d_{G1}(P1)\} + d_{AB}$, after considering the time spent on gates A and B. ∎

2.2 Timing Error Masking with Approximate Logic

Generally speaking, a suspicious FF is driven by multiple paths, and timing errors may occur only when speed-paths are sensitized. In other words, timing errors may be activated by only a few minterms of the truth table for a suspicious FF, denoted as *critical minterms*. Motivated by this observation, according to *Theorem 1*, if all the critical minterms are covered with approximate logic, we can achieve large timing slack and mask potential timing errors. *The question now becomes how to efficiently construct redundant approximate logic for speed-paths?*

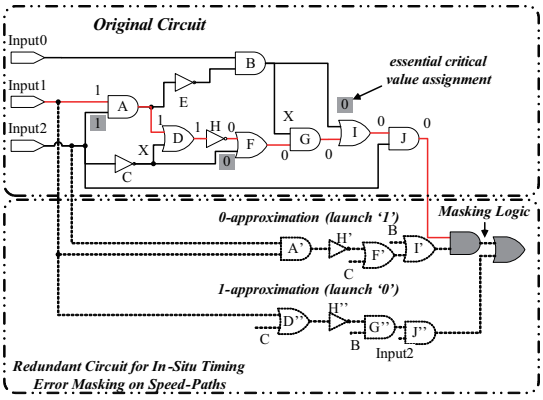


Figure 3: Speed-Path Approximation.

Let us consider an example circuit shown in Fig. 3 for explanation, wherein path P $\{Input1, A, D, H, F, G, I, J\}$ is a speed-path. When logic '1' is applied at $Input1$ and propagates along this path to generate logic '0' at the receiving end, we have to assign logic '1' at the side-input[3] of gate A. This is because, logic '1' is a non-controlling value of *AND* gate, and the output of A will be dominated by the side-input if it is assigned with controlling value. Similarly, side-inputs of gate F and gate I have to be assigned as non-controlling values (see Fig. 3).

Let us define such side-inputs on the path that need to have deterministic non-controlling values (marked in shade) as *essential side-inputs*. Based on the path sensitization theory in [23], we have the following lemma to functionally activate a speed-path:

LEMMA 2. *To cover all the critical minterms that sensitize a particular speed-path is equivalent to approximate its essential side-inputs.*

With the above, we can construct redundant approximate logic for each speed-path by simple structural analysis. Again, take path P in Fig. 3 as an example. Suppose we would like to construct 0-approximate logic for this path, we first duplicate the entire path and then gradually remove those gates without essential side-inputs. To be specific, the removing process is conducted structurally by analyzing the targeted speed-path P reversely from the ending gate (i.e., gate G) to the sending gate (i.e., gate A). Consider gate J, since it is dominated by its on-input with controlling value 0, we can remove it from the approximate logic. Similarly, gate G and gate D are not needed. While the outputs of gates F and A are determined by both on-input and side-input signals, two gates need to be duplicated in the 0-approximate logic, and the side-inputs are connected to the same net as path P in the original circuit. As shown in Fig. 3, the 0-approximate logic constructed as above will output logic '0' if and only if the speed-path P in the original circuit is sensitized with launching value logic '1'. The 1-approximate logic for speed-path P can be constructed similarly (see Fig. 3). Since the approximate logic is with much simpler logic structure and the delay of the masking logic is usually insignificant (without necessarily sizing it up), we can achieve large timing slack and mask potential timing errors on speed-paths.

Note that, not all kinds of logic cells have controlling values for their inputs, e.g., XOR/XNOR gate. If a speed-path contains such kind of logic cells, their side-inputs will be treated as essential side-inputs to have deterministic values to approximate and the proposed methodology is applicable to such designs.

3. INTIMEFIX SYNTHESIS

Adding redundant approximate logic facilitates to achieve timing error resilience on speed-paths. As the construction of the approximate logic needs to take side-input signals from the original circuit, however, two potential problems arise: (i) the latencies for side-inputs may become a concern for the propagation delay of the approximate logic and such side-inputs are denoted as *critical side-inputs* (formally defined later); (ii) the increased loading capacitance on side-inputs can prolong the delay of those paths going through them. The above observations motivate us to propose a cost-efficient and scalable synthesis framework for *InTimeFix*, as illustrated in Fig. 4.

[3]Given a path P = $\{G_0, G_1, ...G_m\}$, for a specific gate G_i, G_{i-1} is the *on-input* signal of G_i, while other input signals of G_i are its *side-inputs*.

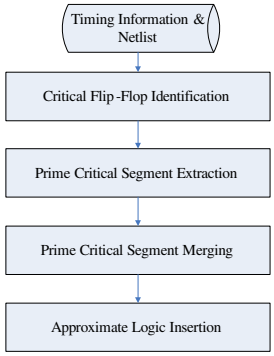


Figure 4: *InTimeFix* Synthesis Framework.

3.1 Overall Flow

Fig. 4 describes the synthesis flow of InTimeFix. With the optimized circuit netlist and the corresponding timing information in standard delay format (SDF), we firstly identify those suspicious FFs that are driven by speed-paths and thus need to be considered. Speed-paths are defined as those paths whose propagation delays exceed a threshold value, e.g., 80% of the maximum path delay. Note that, although static timing analysis is not able to output accurate timing values, its accuracy is sufficient for suspicious FF identification because we only need a comparative relationship among paths and we can always tune the threshold to tradeoff between the hardware cost and protection strength.

Next, a set of so-called *prime critical segments* is extracted from speed-paths, defined as the segments of speed-paths that do not include any critical side-inputs, with which approximate logic can be safely generated with more timing slack. Then, a heuristic method is used to merge prime critical segments to minimize the hardware cost of the approximate logic and the extra loading capacitance of side-inputs. Finally, approximate logic is generated for the merged prime critical segments and inserted into the original circuit for timing error correction.

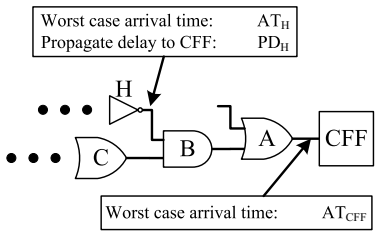


Figure 5: Example to Illustrate the Critical Side-Input.

3.2 Prime Critical Segment Extraction

Before introducing the details of our proposed algorithm, let us first formally define the criteria used to identify critical side-input, as depicted in Fig. 5. Taking side-input H as an example, when considering the critical flip-flop CFF, we have the worst case arrival time AT_H on side-input H, the propagation delay PD_H from H to CFF, and the worst case arrival time AT_{CFF} on CFF. Suppose RD is a predefined reduced delay (i.e., extra slack) that we want to achieve and MD is the delay of the masking logic, then H is a critical side-input if $AT_{CFF} - (AT_H + PD_H + MD) > RD$. The basic idea behind this definition is that, if this criteria is not satisfied, there is sufficient timing slack on this side-input and it does not affect the timing of the approximate logic at all. Based on the above definition, we denote a gate on speed-path to be prime critical gate if all its side-inputs are non-critical. Furthermore, a segment of a speed-path is a prime critical segment if it only consists of prime critical gates.

Our prime critical segment extraction algorithm (denoted as *ExPriSeg*) is shown in Algorithm 1, where, *Gate* is one gate on a speed-path and *Segment* is the parameter to store the targeted segment; *SegmentSet* denotes the set of extracted prime critical segments; while *Ogate* and *Cside* represent the on-input and critical side-inputs of *Gate*.

Algorithm 1: Extract Prime Critical Segment(ExPriSeg)

```
   input: Gate,Segment
1  begin
2      if PI or FF is reached then
3          if Segment is not illegal then
4              return FailToExtract;
5          else
6              add Segment into SegmentSet;
7              return SucceedToExtract;
8      if Gate is prime critical gate then
9          add Gate into Segment;
10         find the on-input Ogate;
11         return ExPriSeg(Ogate,Segment);
12     else
13         if Segment is legal then
14             add Segment into SegmentSet;
15             return SucceedToExtract;
16         else
17             foreach Cside do
18                 new Segment;
19                 if ExPriSeg(Cside,Segment)==FailToExtract then
20                     return FailToExtract;
21             return SucceedToExtract;
```

To relax the timing slack for a critical flip-flop by *RD*, we need to reduce the delay of all the speed-paths connected to it. Here we regard an extracted prime critical segment as a *legal* one if the approximate logic for it is able to reduce the delay of the targeted speed-path for at least *RD*. Starting from a specific critical flip-flop, our algorithm extracts the prime critical segments by recursively tracing its fan-in cone in a depth-first manner. Initially, *Segment* is empty and *Gate* is the input gate of the targeted critical flip-flop. During the tracing procedure, once a primary input or a flip-flop, denoted as *PI* or *FF*, is reached (Line 2), function returns *FailToExtract* if there is no legal prime critical segment found, otherwise we keep the *Segment* and return *SucceedToExtract*. Suppose *Gate* is detected to be a prime critical gate, we add *Gate* into *Segment* and keep on tracing its on-input gate. On the other hand, if *Gate* is not a prime critical gate (Line 12), we first check whether the current *Segment* is legal or not. The searching process is stopped by storing *Segment* and return *SucceedToExtract* if *Segment* is legal prime critical segment, otherwise, we empty the current *Segment* and start to trace each critical side-input separately. Clearly, the extracted prime critical segments can cover all the speed-paths ending at the targeted flip-flop if the final returned value is *SucceedToExtract*. Suppose it returns *FailToextract*, we will try to reduce the value of *RD* by a pre-defined ratio, and conduct the same search again. One thing to note is that our method tends to extract legal prime critical segments that are close to the targeted flip-flops, and they are able to cover more speed-paths, if any.

3.3 Prime Critical Segment Merging

The prime critical segments extracted from different critical flip-flops are likely to be merged to further reduce hardware cost. As the example shown in Fig. 6, two prime critical segments $\{E,D,...C,A\}$ and $\{E,D,...C,B\}$ can share a merged prime critical segment $\{E,D,...C\}$. Furthermore, we notice that our extracted prime critical segments are not in the most compact format. Taking segment $\{E,D,...C,A\}$ as an example, it is not necessary to approximate the entire segment, instead, only by approximating $\{E,D,...C\}$ is enough to achieve *RD* delay reduction. We denote the length (i.e., the number of logic gates) of the shortest legal subpart of prime critical segment as the *essential length*, and represent the length of the remaining part as the *redundant length*. For the sake of simplicity, we regard a subpart of a prime critical segment still legal if the length of the subpart is larger than the essential length. Therefore, two prime critical segments can be merged if the length of the shared part is larger than both of their essential lengths.

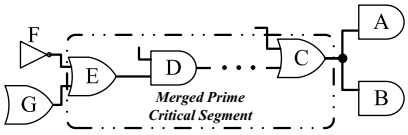


Figure 6: Prime Critical Segment Merging: An Example.

The main flow of our proposed prime critical segment merging algorithm is shown in Fig. 7, comprising the following steps:

1. Starting from the extracted prime critical segment set, we first sort them in non-decreasing order in terms of their redundant length. The basic idea behind this step is that we need to first fix those prime critical segment with less flexibility. Then, all the segments are merged by iteratively applying the following steps.

2. We always select the top un-processed segment and employ a heuristic method to find the optimal subpart. First of all, the closest-to-output gate on the selected segment shared by most remaining un-processed segments is identified. Taking the circuit shown in Fig. 6 as example, gate C is picked first if the selected segment is $\{E,D,C,A\}$ since it has higher probability to replace the most remaining un-processed segments by backwardly tracing the selected segment from this gate, say, segment $\{E,D,C\}$. Suppose that the length of $\{E,D,C\}$ is less than the essential length, we extend this sub-segment to $\{E,D,C,A\}$ and such extension is conducted until the length requirement is satisfied. Suppose the length of $\{E,D,C\}$ is larger than the essential length, we enumerate all the possible legal subparts to identify the one that includes minimal number of side-inputs in the hope that increased loading capacitance is minimized when inserting approximate logic. Finally, the optimal segment is fixed.

3. The remaining un-processed segments are checked to determine whether they can be replaced by newly fixed segment. The replaceable segments are labeled as processed.

4. The procedure terminates if all the segments have been processed, otherwise it goes back to step 2.

4. EXPERIMENTAL RESULTS

4.1 Experimental Setup

To evaluate the effectiveness of our proposed *InTimeFix* technique, we conduct experiments on two large ISCAS'89 benchmark circuits, *s38417* and *s38584*, as well as three large IWLS benchmark circuits, *wb_conmax*, *ethernet* and *des_perf*.

In the experimental flow, we first synthesize the benchmark circuits with Synopsys Design Compiler to obtain the optimized circuit netlist and its SDF timing information. They are then fed to *InTimeFix* to generate redundant approximate logic to mask potential timing errors, targeting speed-paths within 20% of the longest path delay. Finally, Synopsys PrimeTime is applied to evaluate the solution, and the quality of the solution is demonstrated by the extra timing slack achieved with the new circuit when compared with the original one.

4.2 Experimental Results

When comparing the worst case delay (WCD) between the original circuit and the one equipped with redundant approximate logic (see Table 1, Columns 7-10), it can be observed that the proposed solution is able to achieve 11.15% timing slack relaxation on average. On the other hand, as shown in Columns 5-6, the hardware cost (the unit of cost is the area of smallest 2 input *AND* gates) introduced in the proposed *InTimeFix* technique to achieve the above timing slack is extremely low, less than 0.89% on average. As can be seen from Column 11, the runtime to process the largest benchmark circuit *ethernet* takes less than one second. Consequently, we believe the proposed methodology can be easily scalable to large industrial designs.

Benchmark	Circuit Size (# of gates)	FF (#)	Critical FF (#)	Cost (# of gates)	Increased Ratio (%)	Ori. WCD (ns)	Our WCD (ns)	Relaxed Slack (ns)	Improved Ratio (%)	Runtime (s)
s38417	24370	1636	78	570	2.34	35.34	31.93	3.41	9.64	0.09
s38584	21066	1426	12	98	0.47	20.50	18.49	2.01	9.80	0.1
des_perf	154323	9105	89	592	0.38	7.80	6.68	1.12	14.36	0.95
wb_conmax	75352	3316	277	1160	1.54	8.47	7.74	0.73	8.59	0.4
ethernet	157841	10752	28	386	0.24	8.21	7.11	1.10	13.38	1.083
Ave.					0.89				11.15	

Table 1: Experimental Results on Improved Timing Slack and Hardware Cost.

Benchmark	Gate Sizing with Area Constraint vs. *InTimeFix*				*InTimeFix* on top of Gate Sizing				
	Area Constraint (# of gates)	Gate Sizing WCD (ns)	*InTimeFix* WCD (ns)	Improvement (%)	Gate Sizing WCD (ns)	Area Cost (# of gates)	*InTimeFix* WCD (ns)	Area Cost (# of gates)	Further Improvement (%)
s38417	570	34.70	31.93	7.99	31.93	1050	27.90	608	12.63
s38584	98	19.96	18.49	7.36	15.18	1217	13.85	171	8.81
des_perf	592	6.95	6.68	3.88	6.43	5937	6.06	1309	5.85
wb_conmax	1160	7.47	7.74	-3.61	5.38	4008	4.88	1561	9.42
ethernet	386	7.23	7.11	1.73	6.15	6257	5.25	614	14.61
Ave.				3.47					10.26

Table 2: Comparison on Timing Slack Improvement: Gate Sizing vs. *InTimeFix*.

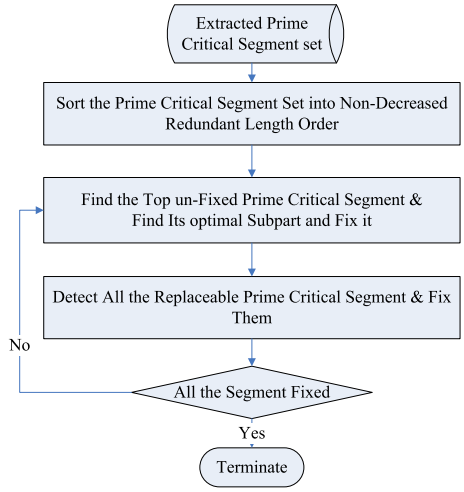


Figure 7: Flowchart of Prime Critical Segment Merging.

A close examination of the experimental results show that benchmark circuits *s*38417 and *wb_conmax* consume the largest percentage of hardware overhead. The reason is that the speed-paths in these two benchmarks are quite evenly distributed and the prime critical segments identified from different critical flip-flops are more likely to be independent to each other. Therefore, the hardware cost is higher than other benchmark circuits. To have a better design tradeoff, we try to reduce the hardware cost for these two benchmarks by gradually removing the approximation logic on the shortest speed-paths. Even when the hardware cost is reduced to less than 1%, we can still achieve 7.1% and 11.6% slack relaxation for *wb_conmax* and *s*38417, respectively.

With the above experimental results, we can observe clear advantages of the proposed *InTimeFix* technique over [16], without performing direct comparisons. In [16], the authors conducted experiments on a number of benchmark circuits whose sizes are less than 2000 gates. Their experimental results show that, on average 18% area overhead is required to mask timing errors on speed-paths within 10% of the longest path delay (the minimum overhead is 4%).

While not targeting timing error resilience, gate sizing is an effective technique to improve circuit timing. In our next experiment, we compare our *InTimeFix* solution against a greedy gate sizing technique [19]. Firstly, when we constrain the area overhead for gate sizing solution to be the same as the one with *InTimeFix* in earlier experiment, it can be seen that *InTimeFix* outperforms gate sizing in most cases (except *wb_conmax*), and the average improvement is 3.47%, which proves the cost-efficiency of the proposed solution. Next, since *InTimeFix* is compatible with gate sizing technique, we combine the two solutions in such manner that we first employ gate sizing to improve circuit timing until no further benefits can be achieved and then apply *InTimeFix* on top of it. We can observe that, without area constraints, gate sizing can significantly improve circuit performance, but at considerable area and power cost. *InTimeFix* is able to provide additional 10.26% timing slack on average, and the area cost is still quite small.

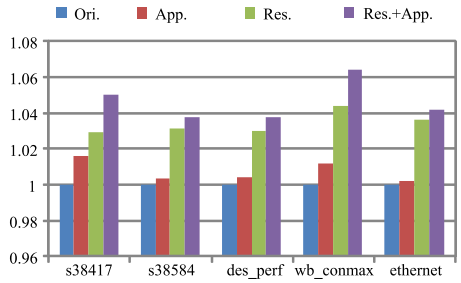
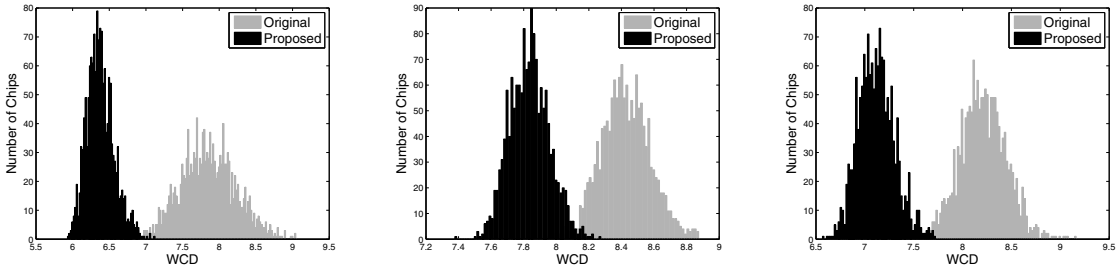


Figure 8: Power Comparison Results.

Fig. 8 compares the power consumption of the following circuit netlists: (i) original circuits denoted as *"Ori."*; (ii) original circuits equipped with *InTimeFix* logic, denoted as *"App."*; (iii) circuits after gate sizing, denoted as *"Res."*; (iv) resized circuits equipped with *InTimeFix* logic, denoted as *"Res.+App."*. The power values are obtained with Synopsys PrimeTIme PX tool and normalized with respect to that of original circuit. From this figure, we can observe the power impact of applying *InTimeFix* is not significant and vary with different circuits, on average about 1-2%. Another observation is that the power impact of applying *InTimeFix* with a resized circuit is usually higher than applying it with the original circuit for the same circuit. This is because critical paths are more balanced after gate sizing and we need to approximate more circuit speed-paths.

Finally, we evaluate the impact of process variation on the proposed *InTimeFix* architecture, using Monte Carlo simulation. According to [21], we assume there is 10% variation on each standard cell. The results are depicted in Fig. 9, where we plot and compare the WCD distributions of two sets of circuits (i.e. the black pile represents set of processed circuits and the gray one denotes the original set of circuits) for the three large IWLS circuits. As can be seen from the figure, even for circuit with *InTimeFix* approximate logic under the worst case process variation corner, it has smaller or similar WCD when comparing with that of the original circuit under the best case scenario. Moreover, it can be observed that the number of the closer-to-mean chips increases and the standard deviation of WCD distribution shrinks with *InTimeFix*. In particular, as can be observed in Fig. 9(a), benchmark circuit *des_perf* with *InTimeFix* has only one third of the distribution width when compared to that of the original circuit.

The above phenomenon demonstrate that the proposed *InTimeFix* technique facilitates to tolerate process variation effects. The reason is that our proposed method effectively reduces the number of logic elements on the critical paths and also shrinks the variation, behind which the mathematic principle can be explained by the example given in [22]: assuming inverters have independent gaussian delay distribution (μ, σ), the delay of a path including n inverters obey the gaussian distribution $(n\mu, \sqrt{n}\sigma)$. Clearly, less n leads to smaller deviation.



(a) des_perf (b) wb_conmax (c) ethernet

Figure 9: Circuit Timing under Process Variation

5. DISCUSSIONS

5.1 Limitations of InTimeFix

The effectiveness of *InTimeFix* is highly related to the original circuit structure with its current synthesis flow. In this section, we discuss certain circuit structure that are difficult to be optimized with *InTimeFix*. As shown in *Lemma 2*, for a particular speed-path, we need to approximate its essential side-inputs when generating the redundant approximation circuit. Consequently, if the number of essential side-inputs is large, the timing slack provided by the approximation circuit delay would be small. In the extreme case (e.g., to generate 1-approximation for a speed-path that contains nothing but a series of *AND* gates), the redundant circuit may simply be a duplication of the speed-path, rendering *InTimeFix* not effective at all.

To mitigate the above problems, it would be beneficial to manipulate the original circuit structure for *InTimeFix*. That is, we could synthesize the circuit in a *InTimeFix-friendly* manner so that speed-paths are easy to approximate, whenever possible. Alternatively, we could resort to circuit re-synthesis techniques such as retiming and/or rewiring to make the circuit more *InTimeFix-friendly*. We are currently investigating the above techniques to mitigate the limitations of *InTimeFix*.

5.2 Testing Circuits with InTimeFix

From manufacturing test perspective, adding redundancy into the circuit may introduce untestable faults, and there is no exception for our design. For example, for the circuit shown in Fig. 3, the stuck-at-1 fault at the output of $G0$ is untestable, because to activate this fault, we need to set it as logic '0', but when $G0 = 0$, the original circuit F will also output logic '0', preventing the propagation of its faulty effect to outputs. Similarly, the stuck-at-0 fault at the output of $G1$ is untestable. It is important to note that, the presence of such faults would not affect the functional correctness of the circuit. At the same time, they do render the corresponding timing error masking logic to be ineffective, but the likelihood for such faults is quite low due to their small sizes.

One way to make these faults testable is to add some design-for-testability (DfT) circuit, e.g., adding additional flip-flop to be driven by G0/G1 directly. This, however, increases the hardware cost and also prolongs the propagation delay on approximate logic paths due to extra load. Alternatively, we can rely on delay testing to guarantee the timing correctness of the corresponding speed-paths. In other words, as long as the path can pass at-speed test, its timing correctness is guaranteed and we do not need to care whether these untestable faults exit or not.

6. CONCLUSION

The timing behavior of integrated circuits has become increasingly uncertain with technology scaling. In this paper, we propose the so-called *InTimeFix* technique to achieve low-cost and scalable timing error resilience in logic circuits, by introducing fine-grained redundant *approximate logic* for speed-paths in the circuit. As the proposed synthesis methodology for the redundant circuit only relies on simple structural analysis of the original circuit, it can be easily scalable to large IC designs. Experimental results demonstrate that the proposed solution can effectively increase circuit timing slack with very low cost. In addition, as a redundancy scheme, *InTimeFix* can be combined with other timing optimization techniques (e.g., gate sizing) to further improve circuit performance under variation.

7. ACKNOWLEDGEMENT

This work was supported in part by the Hong Kong SAR Research Grants Council under General Research Fund No. CUHK418111 and No. CUHK418812.

8. REFERENCES

[1] S. Borkar. Designing reliable systems from unreliable components: the challenges of transistor variability and degradation. *IEEE Micro*, 25(6):10–16, 2005.

[2] K. Bowman, et al. Circuit techniques for dynamic variation tolerance. In *Proc. DAC*, pp. 4–7, 2009.

[3] D. Frank, R. Puri, and D. Toma. Design and CAD Challenges in 45nm CMOS and beyond. In *Proc. ICCAD*, pp. 329–333, 2006.

[4] T. Austin and V. Bertacco. Deployment of better than worst-case design: solutions and needs. In *Proc. ICCD*, pp. 550–555, 2005.

[5] M. Agarwal, B. C. Paul, M. Zhang, and S. Mitra. Circuit Failure Prediction and Its Application to Transistor Aging. In *Proc. VTS*, pp. 277–286, 2007.

[6] D. Ernst, et al. Razor: a low-power pipeline based on circuit-level timing speculation. In *Proc. MICRO*, pp. 7–18, 2003.

[7] K. Bowman, et al. Energy-efficient and metastability-immune resilient circuits for dynamic variation tolerance. *IEEE JSSC*, 44(1):49–63, 2009.

[8] S. Das, et al. RazorII: In situ error detection and correction for PVT and SER tolerance. *IEEE JSSC*, 44(1):32–48, 2009.

[9] B. Greskamp, et al. Blueshift: Designing processors for timing speculation from the ground up. *Proc. HPCA*, pp. 213-224, 2009.

[10] R. Ye, F. Yuan and Q. Xu. Online clock skew tuning for timing speculation. *Proc. ICCAD*, pp. 442–447, 2011.

[11] Y. Liu, et al. On logic synthesis for timing speculation. *Proc. ICCAD*, pp. 591–596, 2012.

[12] R. Ye, et al. Post-placement voltage island generation for timing-speculative circuits. *Proc. DAC*, 2013.

[13] M. R. Choudhury and K. Mohanram. TIMBER: Time borrowing and error relaying for online timing error resilience. In *Proc. DATE*, pp. 1554–1559, 2010.

[14] M. Kurimoto, et al. Phase-adjustable error detection flip-flops with 2-stage hold driven optimization and slack based grouping scheme for dynamic voltage scaling. In *Proc. DAC*, pp. 884–889, 2008.

[15] K. Hirose, et al. Delay-compensation flip-flop with in-situ error monitoring for low-power and timing-error-tolerant circuit design. *Japan Journal of Applied Physics*, 47(4):2779–2787, Apr. 2008.

[16] M. R. Choudhury and K. Mohanram. Masking timing errors on speed-paths in logic circuits. In *Proc. DATE*, pp. 87–92, 2009.

[17] S.-L. Lu. Speeding up processing with approximation circuits. *Computer*, 37(3):67–73, Mar. 2004.

[18] M. R. Choudhury and K. Mohanram. Approximate logic circuits for low overhead, non-intrusive concurrent error detection. In *DATE*, pp. 903–908, 2008.

[19] O. Coudert, *et al.*, New Algorithm for Gate Sizing: A Comparative Study. In *Proc. DAC*, pp. 734-739, 1996.

[20] M. Mani, *et al.*, An Efficient Algorithm for Statistical Minimization of Total Power under Timing Yield Constraints. In *Proc. DAC*, pp. 309-314, 2005.

[21] G. Yu, et al. Statistical Static Timing Analysis Considering Process Variation Model Uncertainty. In *Proc. IEEE TCAD*, pp. 1880-1890, 2008.

[22] J. L. Tsai, et al. A Yield Improvement Methodology Using Pre- and Post-Silicon Statistical Clock Scheduling. In *Proc. ICCAD*, pp. 611-618, 2004.

[23] K.-T. Cheng and H.-C. Chen, Classification and Identification of Nonrobust Untestable Path Delay Faults. In *IEEE TCAD*, 15(8):845-853, Aug. 1996.